house_prices1

November 11, 2021

1 Predicting house prices (part 1)

To test techniques for supervised learning discussed so far we train a model for predicting house prices in Germany. Inputs are properties of a house and of the plot of land it has been built on. Output is the selling price.

Training data exists in form of advertisements on specialized websites for finding a buyer for a house. In principle we could scrape data from such a website, but usually its not allowed by the website operator and we would have to write lots of code. Erdogan Seref already did this job at www.immobilienscout24.de and published the data set at www.kaggle.com under a Attribution-NonCommercial-ShareAlike 4.0 International License.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as linear_model
import sklearn.metrics as metrics
import sklearn.model_selection as model_selection
import sklearn.preprocessing as preprocessing
import sklearn.pipeline as pipeline

data_path = 'house_prices1/'
```

1.1 The data set

At first we load the data set and try to get an overview of features and data quality.

[2]:
```
data = pd.read_csv(data_path + 'german_housing.csv')
```

If a data frame has many columns Pandas by default does not show all columns. But we want to see all. Thus, we have to adjust the number of columns shown by setting corresponding option to `None` (that is, unlimited).

[3]:
```
pd.set_option('display.max_columns', None)
data.head(10)
```

[3]:

	Unnamed: 0	Price	Type	Living_space	Lot
0	0	498000.0	Multiple dwelling	106.00	229.0
1	1	495000.0	Mid-terrace house	140.93	517.0
2	2	749000.0	Farmhouse	162.89	82.0
3	3	259000.0	Farmhouse	140.00	814.0
4	4	469000.0	Multiple dwelling	115.00	244.0
5	5	1400000.0	Mid-terrace house	310.00	860.0
6	6	3500000.0	Duplex	502.00	5300.0
7	7	630000.0	Duplex	263.00	406.0
8	8	364000.0	Duplex	227.00	973.0
9	9	1900000.0	Duplex	787.00	933.0

	Usable_area	Free_of_Relation	Rooms	Bedrooms	Bathrooms	Floors
0	NaN	01.10.2020	5.5	3.0	1.0	2.0
1	20.00	01.01.2021	6.0	3.0	2.0	NaN
2	37.62	01.07.2020	5.0	3.0	2.0	4.0
3	NaN	nach Vereinbarung	4.0	NaN	2.0	2.0
4	NaN	sofort	4.5	2.0	1.0	NaN
5	100.00	sofort	8.0	NaN	NaN	3.0
6	163.16	nach Absprache	13.0	NaN	4.0	NaN
7	118.00	01.04.2020	10.0	NaN	NaN	3.0
8	83.00	nach Absprache	10.0	4.0	4.0	2.0
9	NaN	NaN	30.0	NaN	NaN	3.0

	Year_built	Furnishing_quality	Year_renovated	Condition
0	2005.0	normal	NaN	modernized
1	1994.0	basic	NaN	modernized
2	2013.0	NaN	NaN	dilapidated
3	1900.0	basic	2000.0	fixer-upper
4	1968.0	refined	2019.0	refurbished
5	1969.0	basic	NaN	maintained
6	2004.0	basic	NaN	dilapidated
7	1989.0	basic	NaN	modernized
8	1809.0	normal	2015.0	modernized
9	1920.0	basic	NaN	modernized

	Heating	Energy_source	Energy_certificate
0	central heating	Gas	available
1	stove heating	NaN	not required by law
2	stove heating	Fernwärme, Bioenergie	available
3	central heating	Strom	available
4	central heating	Öl	available
5	NaN	Öl	available
6	stove heating	Erdwärme, Holzpellets	available

```
7    stove heating                    Gas             available
8  central heating                  Strom             available
9    stove heating    Gas, Fernwärme-Dampf             available

   Energy_certificate_type  Energy_consumption Energy_efficiency_class  \
0       demand certificate                 NaN                       D
1                      NaN                 NaN                     NaN
2       demand certificate                 NaN                       B
3       demand certificate                 NaN                       G
4       demand certificate                 NaN                       F
5  consumption certificate                 NaN                     NaN
6  consumption certificate                35.9                       A
7       demand certificate                 NaN                       E
8  consumption certificate               183.1                       F
9       demand certificate                 NaN                       D

               State                  City                    Place  Garages  \
0  Baden-Württemberg         Bodenseekreis              Bermatingen      2.0
1  Baden-Württemberg      Konstanz (Kreis)                    Engen      7.0
2  Baden-Württemberg     Esslingen (Kreis)                Ostfildern      1.0
3  Baden-Württemberg      Waldshut (Kreis)  Bonndorf im Schwarzwald      1.0
4  Baden-Württemberg     Esslingen (Kreis)  Leinfelden-Echterdingen      1.0
5  Baden-Württemberg             Stuttgart                      Süd      2.0
6  Baden-Württemberg     Göppingen (Kreis)                   Wangen      7.0
7  Baden-Württemberg  Freiburg im Breisgau                Munzingen      2.0
8  Baden-Württemberg              Enzkreis                Neuenbürg      8.0
9  Baden-Württemberg              Mannheim                  Rheinau     12.0

    Garagetype
0  Parking lot
1  Parking lot
2       Garage
3       Garage
4       Garage
5       Garage
6  Parking lot
7       Garage
8  Parking lot
9  Parking lot
```

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10552 entries, 0 to 10551
Data columns (total 26 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Unnamed: 0              10552 non-null  int64
```

```
 1   Price                    10552 non-null  float64
 2   Type                     10150 non-null  object
 3   Living_space             10552 non-null  float64
 4   Lot                      10552 non-null  float64
 5   Usable_area              5568 non-null   float64
 6   Free_of_Relation         6983 non-null   object
 7   Rooms                    10552 non-null  float64
 8   Bedrooms                 6878 non-null   float64
 9   Bathrooms                8751 non-null   float64
 10  Floors                   7888 non-null   float64
 11  Year_built               9858 non-null   float64
 12  Furnishing_quality       7826 non-null   object
 13  Year_renovated           5349 non-null   float64
 14  Condition                10229 non-null  object
 15  Heating                  9968 non-null   object
 16  Energy_source            9325 non-null   object
 17  Energy_certificate       9797 non-null   object
 18  Energy_certificate_type  7026 non-null   object
 19  Energy_consumption       2433 non-null   float64
 20  Energy_efficiency_class  5733 non-null   object
 21  State                    10551 non-null  object
 22  City                     10551 non-null  object
 23  Place                    10262 non-null  object
 24  Garages                  8592 non-null   float64
 25  Garagetype               8592 non-null   object
dtypes: float64(12), int64(1), object(13)
memory usage: 2.1+ MB
```

We should drop irrelevant columns and adjust data types.

- `Unnamed: 0`: Seems to be an integer index. We don't need it, so drop it.
- `Price`: This is our target variable.
- `Type`: An important column, because house prices are likely to depend on the type of house. We should convert this to categorical type.
- `Living_space` and `Lot`: Important features, keep them.
- `Usable_area`: Likely to have influence on the selling price, but available only for half the samples. If we want to use this for regression, we would have to drop half the training samples. Alternatively we could impute values, but it's very hard to guess usable area from other features. We should drop the column.
- `Free_of_Relation`: Not related to the selling price. Drop it.
- `Rooms`, `Bedrooms`, `Bathrooms`: Should have influence on prices, but not available for all samples. For the moment we keep all three columns. Later we should have a look on correlations between the three columns and possibly only keep the first one, which is available for all samples.
- `Floors`: Important feature, keep it.
- `Year_built`: Important feature, keep it.
- `Furnishing_quality`: Important, convert to categorical and keep.
- `Year_renovated`: Important, but half the data is missing. There is good chance that missing values indicate that there the house has not been renovated until today. Thus, a reasonable

fill value is the year of construction.
- `Condition`: Important, convert to categorical and keep.
- `Heating` and `Energy_source`: Could be important, convert to categorical and keep.
- `Energy_certificate`, `Energy_certificate_type`, `Energy_consumption`: The first contains more or less only the value `'available'` (since energy certificates are required by law). The second is irrelevant and the third is missing for most samples. Drop them all.
- `Energy_efficiency_class`: Likely to have influence on the selling price, although classification procedure is very unreliable in practice. Keep and convert to categorical.
- `State`, `City`, `Place`: Geolocation surely influences selling prices. But it's hard to use location data for regression. For the moment we keep these columns.
- `Garages`: Could be important, keep.
- `Garagetype`: If we keep `Garages` then we also have to keep this column. Convert to categorical and rename to `Garage_type` to fit naming convention used for the other columns.

[5]:
```
data = data.drop(columns=['Unnamed: 0', 'Usable_area', 'Free_of_Relation',
                          'Energy_certificate', 'Energy_certificate_type',␣
 ↪'Energy_consumption'])

data['Type'] = data['Type'].astype('category')
data['Furnishing_quality'] = data['Furnishing_quality'].astype('category')
data['Condition'] = data['Condition'].astype('category')
data['Heating'] = data['Heating'].astype('category')
data['Energy_source'] = data['Energy_source'].astype('category')
data['Energy_efficiency_class'] = data['Energy_efficiency_class'].
 ↪astype('category')
data['Garagetype'] = data['Garagetype'].astype('category')

data = data.rename(columns={'Garagetype': 'Garage_type'})

nan_mask = data['Year_renovated'].isna()
data.loc[nan_mask, 'Year_renovated'] = data.loc[nan_mask, 'Year_built']
```

Categorical columns `Furnishing_quality`, `Condition` and `Energy_efficiency_class` should have a natural ordering, which should be represented by the data type.

[6]:
```
print(data['Furnishing_quality'].cat.categories)
print(data['Condition'].cat.categories)
print(data['Energy_efficiency_class'].cat.categories)
```

```
Index(['basic', 'luxus', 'normal', 'refined'], dtype='object')
Index(['as new', 'by arrangement', 'dilapidated', 'first occupation',
       'first occupation after refurbishment', 'fixer-upper', 'maintained',
       'modernized', 'refurbished', 'renovated'],
      dtype='object')
Index([' A ', ' A+ ', ' B ', ' C ', ' D ', ' E ', ' F ', ' G ', ' H '],
dtype='object')
```

We should rename same categories and sort them as good as possible.

[7]:
```
data['Furnishing_quality'].cat.rename_categories({'luxus': 'luxury'},␣
↪inplace=True)
data['Furnishing_quality'].cat.reorder_categories(['basic', 'normal',␣
↪'refined', 'luxury'], inplace=True)

data['Condition'].cat.reorder_categories(['first occupation',
                                          'first occupation after␣
↪refurbishment',
                                          'as new',
                                          'maintained',
                                          'renovated',
                                          'modernized',
                                          'refurbished',
                                          'by arrangement',
                                          'fixer-upper',
                                          'dilapidated'], inplace=True)

data['Energy_efficiency_class'].cat.rename_categories({' A ': 'A', ' A+ ':␣
↪'A+', ' B ': 'B',
                                                       ' C ': 'C', ' D ': 'D',␣
↪' E ': 'E',
                                                       ' F ': 'F', ' G ': 'G',␣
↪' H ': 'H'}, inplace=True)
data['Energy_efficiency_class'].cat.reorder_categories(['A+', 'A', 'B', 'C',␣
↪'D',
                                                        'E', 'F', 'G', 'H'],␣
↪inplace=True)
```

Now let's see how many complete samples we have.

[8]:
```
len(data.dropna())
```

[8]:
```
1591
```

That's very few. So we should drop some columns with many missing values.

[9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10552 entries, 0 to 10551
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Price                   10552 non-null  float64
 1   Type                    10150 non-null  category
 2   Living_space            10552 non-null  float64
 3   Lot                     10552 non-null  float64
 4   Rooms                   10552 non-null  float64
```

```
 5   Bedrooms                 6878 non-null   float64
 6   Bathrooms                8751 non-null   float64
 7   Floors                   7888 non-null   float64
 8   Year_built               9858 non-null   float64
 9   Furnishing_quality       7826 non-null   category
 10  Year_renovated           10211 non-null  float64
 11  Condition                10229 non-null  category
 12  Heating                  9968 non-null   category
 13  Energy_source            9325 non-null   category
 14  Energy_efficiency_class  5733 non-null   category
 15  State                    10551 non-null  object
 16  City                     10551 non-null  object
 17  Place                    10262 non-null  object
 18  Garages                  8592 non-null   float64
 19  Garage_type              8592 non-null   category
dtypes: category(7), float64(10), object(3)
memory usage: 1.1+ MB
```

`Energy_efficiency_class` is relatively unreliable and not too important for selling prices.

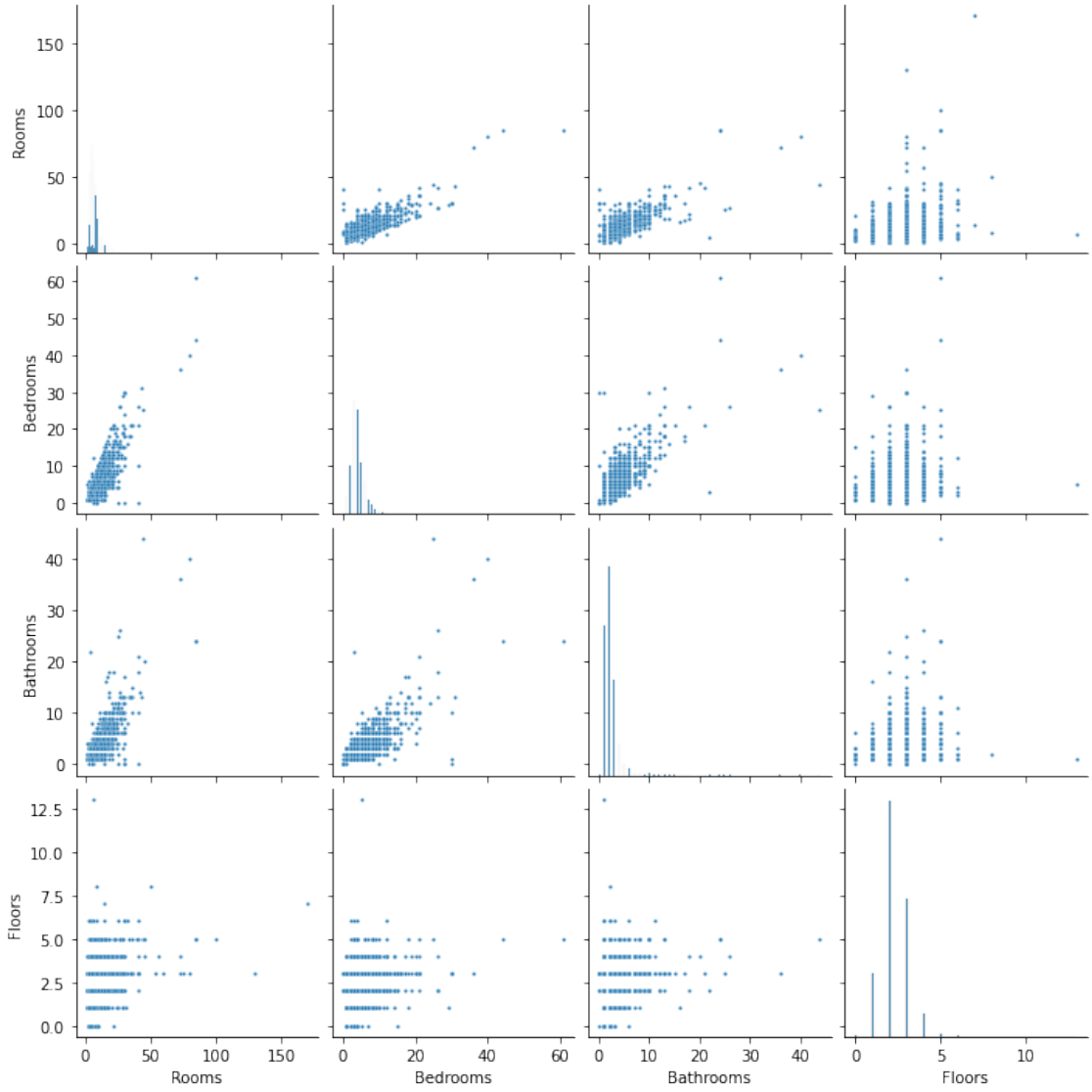
[10]:
```
len(data.drop(columns=['Energy_efficiency_class']).dropna())
```

[10]:
```
2615
```

Better, but not good. The `Bedrooms` column has many missing values, too, and it's likely to be correlated to `Rooms`. So let's look at correlations between `Rooms`, `Bedrooms`, `Bathrooms`, `Floors`.

[11]:
```
sns.pairplot(data[['Rooms', 'Bedrooms', 'Bathrooms', 'Floors']], plot_kws={"s":␣
↪5})
```

[11]:
```
<seaborn.axisgrid.PairGrid at 0x7f08a0c0c890>
```



`Floors` is not correlated to the other columns, so keep it. `Bedrooms` show correlation to `Rooms` and `Bathrooms`, so drop `Bedrooms`. `Bathroom` shows some correlation to `Rooms`. Wether to drop `Bathrooms` should be decided by the increase in sample counts.

[12]:
```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms']).dropna())
```

[12]: 3174

[13]:
```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms', 'Bathrooms']).
↪dropna())
```

[13]: 3479

We should keep `Bathrooms`, because dropping it only yields 300 more samples while neglecting

possibly important information. Note that the number of bath rooms can be regarded as a measure for overall furnishing quality. Thus, there should be some correlation to `Furnishing_quality`.

[14]:
```
data.groupby('Furnishing_quality').mean()['Bathrooms']
```

[14]:
```
Furnishing_quality
basic      2.207496
normal     2.363720
refined    1.826739
luxury     2.778761
Name: Bathrooms, dtype: float64
```

In addition, judging about furnishing quality of a house is highly subjective. Thus, we should drop the column to get more samples without missing data.

[15]:
```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms',␣
↪'Furnishing_quality']).dropna())
```

[15]:
```
4526
```

The `Energy_source` is another candidate for dropping, because has more than 1000 missing values and its influence on selling prices should be rather low.

[16]:
```
for cat in data['Energy_source'].cat.categories:
    print(cat)
```

```
Bioenergie
Erdgas leicht
Erdgas leicht, Erdgas schwer
Erdgas schwer
Erdgas schwer, Bioenergie
Erdgas schwer, Holz
Erdwärme
Erdwärme, Fernwärme
Erdwärme, Gas
Erdwärme, Holzpellets
Erdwärme, Solar
Erdwärme, Solar, Holzpellets, Holz
Erdwärme, Solar, Umweltwärme
Erdwärme, Strom
Erdwärme, Umweltwärme
Fernwärme
Fernwärme, Bioenergie
Fernwärme, Flüssiggas
Fernwärme, Nahwärme, KWK fossil
Fernwärme-Dampf
Flüssiggas
Flüssiggas, Holz
Gas
```

Gas, Bioenergie
Gas, Fernwärme
Gas, Fernwärme-Dampf
Gas, Holz
Gas, Holz-Hackschnitzel
Gas, KWK fossil
Gas, Kohle, Holz
Gas, Strom
Gas, Strom, Holz
Gas, Strom, Kohle, Holz
Gas, Wasserenergie
Gas, Öl
Gas, Öl, Holz
Gas, Öl, Kohle
Gas, Öl, Kohle, Holz
Gas, Öl, Strom
Holz
Holz, Bioenergie
Holz-Hackschnitzel
Holzpellets
Holzpellets, Gas
Holzpellets, Gas, Öl
Holzpellets, Holz
Holzpellets, Holz-Hackschnitzel
Holzpellets, Kohle, Holz
Holzpellets, Strom
Holzpellets, Öl
KWK erneuerbar
KWK fossil
KWK regenerativ
Kohle
Kohle, Holz
Kohle/Koks
Nahwärme
Solar
Solar, Bioenergie
Solar, Erdgas schwer
Solar, Gas
Solar, Gas, Holz
Solar, Gas, Strom
Solar, Gas, Strom, Holz
Solar, Gas, Wasserenergie
Solar, Gas, Öl
Solar, Gas, Öl, Holz
Solar, Holz
Solar, Holz-Hackschnitzel
Solar, Holzpellets
Solar, Holzpellets, Holz

```
Solar, Holzpellets, Strom
Solar, Holzpellets, Öl
Solar, Strom
Solar, Strom, Bioenergie
Solar, Umweltwärme
Solar, Öl
Solar, Öl, Bioenergie
Solar, Öl, Holz
Solar, Öl, Holz-Hackschnitzel
Solar, Öl, Strom
Solar, Öl, Strom, KWK fossil
Strom
Strom, Bioenergie
Strom, Flüssiggas
Strom, Holz
Strom, Holz-Hackschnitzel
Strom, Kohle
Strom, Kohle, Holz
Strom, Umweltwärme
Umweltwärme
Wasserenergie
Windenergie
Wärmelieferung
Öl
Öl, Bioenergie
Öl, Fernwärme
Öl, Holz
Öl, Kohle
Öl, Kohle, Holz
Öl, Strom
Öl, Strom, Holz
Öl, Strom, Kohle, Holz
Öl, Umweltwärme
```

Values are very diverse and hard to preprocess for regression. We would have to convert the column to several boolean columns. In addition, some grouping would be necessary (`Holz` is a subcategory of `Bioenergie` and so on).

[17]:
```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms',␣
↪'Furnishing_quality', 'Energy_source']).dropna())
```

[17]:
```
4854
```

Now we have almost 5000 complete samples. Should be a good compromise between completeness and level of detail.

[18]:
```
data = data.drop(columns=['Energy_efficiency_class', 'Bedrooms',␣
↪'Furnishing_quality', 'Energy_source'])
```

```
data = data.dropna()
```

1.2 Outliers and further preprocessing

Now that we have a cleaned data set we should remove outliers. The simplest method of detecting outliers is to look at the ranges of all feature. With `descibe` we get a first overview for numerical features.

[19]:
```
data.describe()
```

[19]:

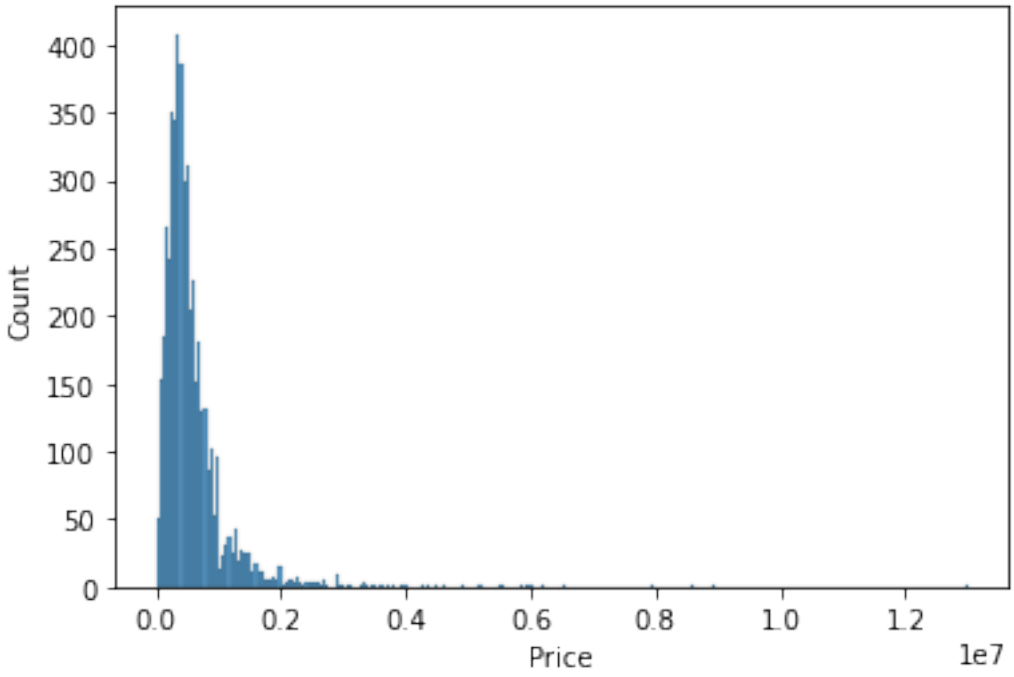
	Price	Living_space	Lot	Rooms	Bathrooms	\
count	4.854000e+03	4854.000000	4854.000000	4854.000000	4854.000000	
mean	5.739566e+05	209.305740	1240.636904	7.051504	2.316028	
std	5.880211e+05	118.252688	3806.518099	3.834865	1.595327	
min	0.000000e+00	0.000000	0.000000	1.000000	0.000000	
25%	2.800000e+05	135.000000	401.000000	5.000000	1.000000	
50%	4.400000e+05	180.000000	675.000000	6.000000	2.000000	
75%	6.850000e+05	248.000000	1042.000000	8.000000	3.000000	
max	1.300000e+07	1742.240000	143432.000000	84.000000	26.000000	

	Floors	Year_built	Year_renovated	Garages
count	4854.000000	4854.000000	4854.000000	4854.000000
mean	2.256696	1964.252369	1995.626700	2.518541
std	0.776769	49.065052	35.389067	2.719901
min	0.000000	1430.000000	1430.000000	1.000000
25%	2.000000	1950.000000	1991.000000	1.000000
50%	2.000000	1974.000000	2008.000000	2.000000
75%	3.000000	1997.750000	2016.000000	3.000000
max	8.000000	2021.000000	2206.000000	65.000000

1.2.1 Price column

[20]:
```
sns.histplot(data['Price'])
```

[20]:
```
<AxesSubplot:xlabel='Price', ylabel='Count'>
```

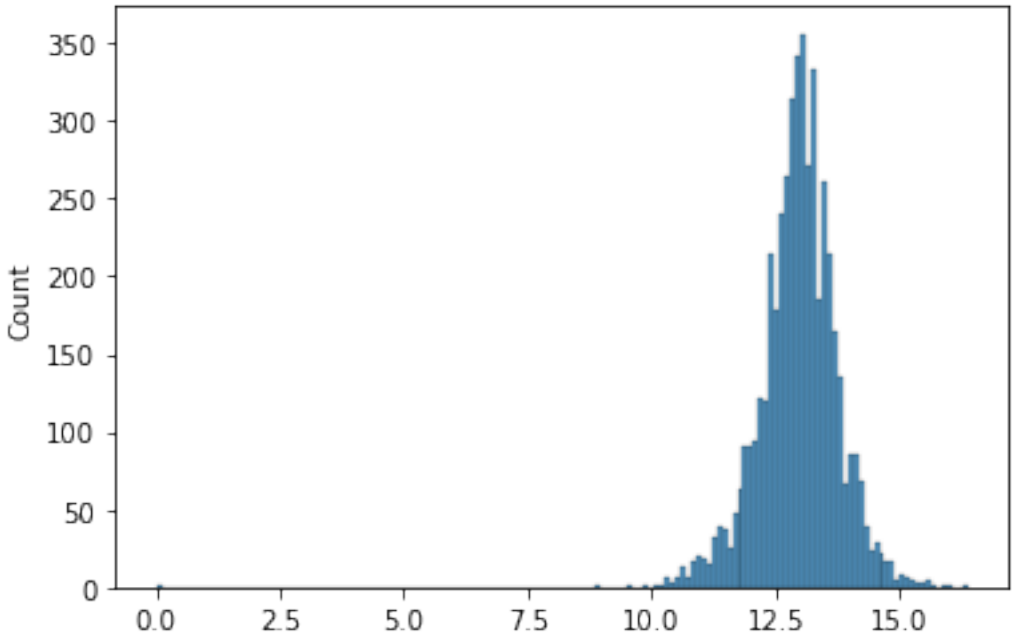


There are only very few high prices and price distribution concentrates on low prices. If the target variable has wide range, but most samples concentrate on a small portion of the range, then 'learning' the target is much more difficult than for more uniformly distributed data.

A common trick is to use nonlinear scaling. Especially for market prices it is known from experience that they follow a log-normal distribution, that is, after applying the logarithm we see a normal distribution. Before applying the logarithm we should samples with zeros at the `Price` column to avoid undefined results. A price of zero indicates that the seller did not provide a price in the advertisement. Thus, dropping such sample even is a good idea if wouldn't want to apply the logarithm.

[21]:
```
data = data.loc[data['Price'] > 0, :]
sns.histplot(np.log(data['Price'].to_numpy()))
```

[21]:
```
<AxesSubplot:ylabel='Count'>
```



There seem to be same very small values.

[22]:
```
np.count_nonzero(np.log(data['Price'].to_numpy()) <= 7)
```

[22]:
```
1
```

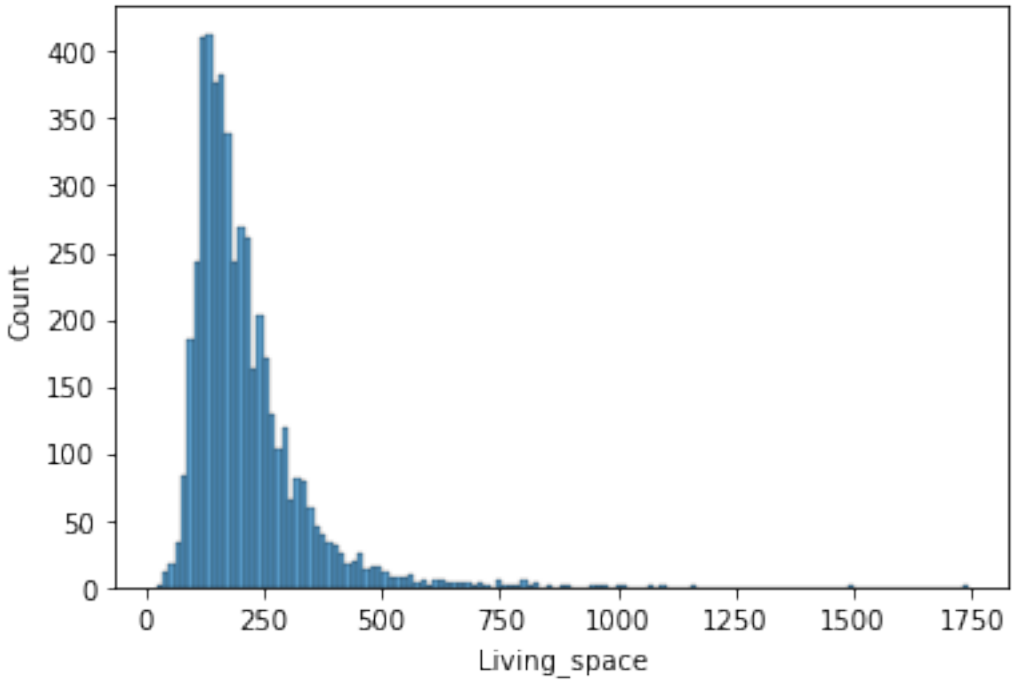
Those 3 samples should be dropped because house prices below $e^7 \approx 1000$ EUR are very uncommon.

[23]:
```
data['Price'] = np.log(data['Price'].to_numpy())
data = data.loc[data['Price'] > 7, :]
```

1.2.2 Living_space column

[24]:
```
sns.histplot(data['Living_space'])
```

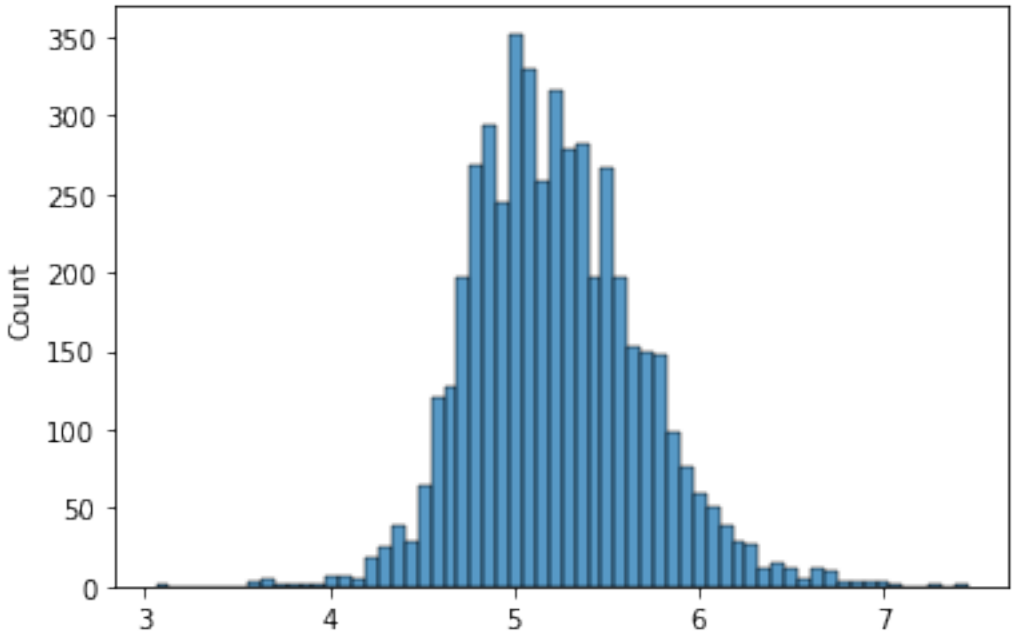
[24]:
```
<AxesSubplot:xlabel='Living_space', ylabel='Count'>
```



Same here as for `Price`.

[25]:
```
data = data.loc[data['Living_space'] > 0, :]
sns.histplot(np.log(data['Living_space'].to_numpy()))
```

[25]:
```
<AxesSubplot:ylabel='Count'>
```



[26]:
```
np.count_nonzero(np.log(data['Living_space'].to_numpy()) <= 1)
```

[26]:
```
0
```

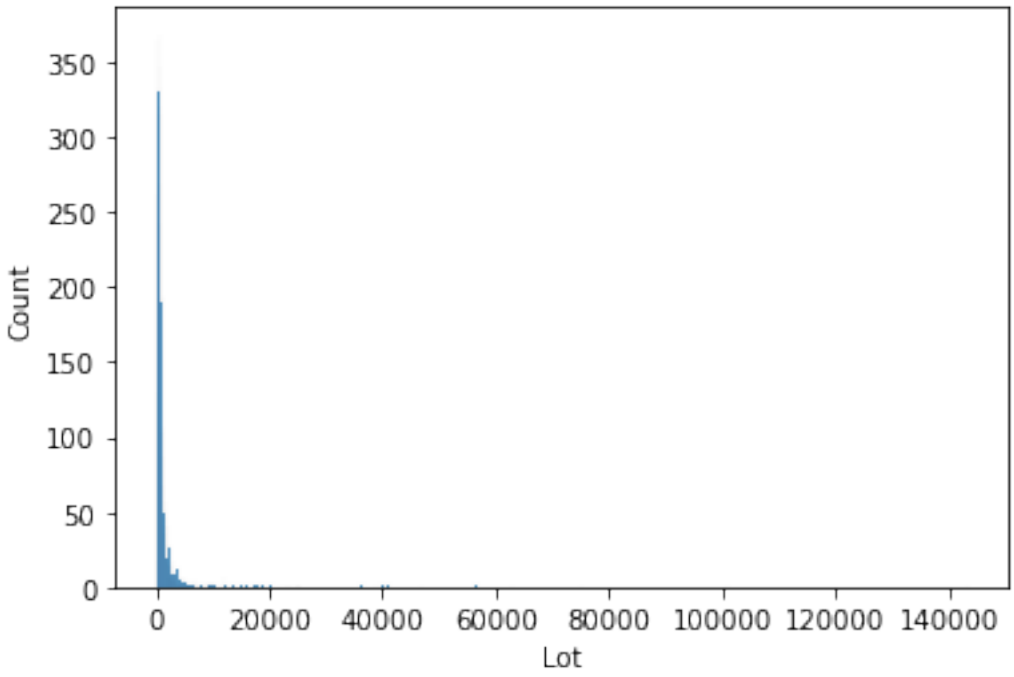
Living space below 3 m^2 is very unlikely.

[27]:
```
data['Living_space'] = np.log(data['Living_space'].to_numpy())
data = data.loc[data['Living_space'] > 1, :]
```

1.2.3 Lot column

[28]:
```
sns.histplot(data['Lot'])
```

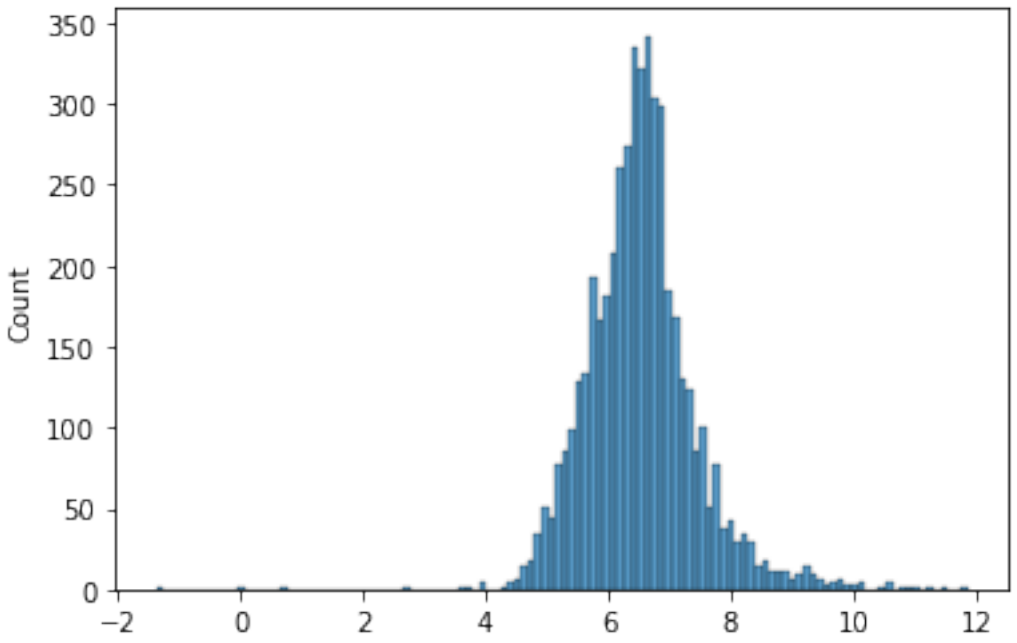
[28]:
```
<AxesSubplot:xlabel='Lot', ylabel='Count'>
```



Same here as for `Price` again.

[29]:
```
data = data.loc[data['Lot'] > 0, :]
sns.histplot(np.log(data['Lot'].to_numpy()))
```

[29]:
```
<AxesSubplot:ylabel='Count'>
```



[30]:
```
np.count_nonzero(np.log(data['Lot'].to_numpy()) <= 4)
```

[30]: 13

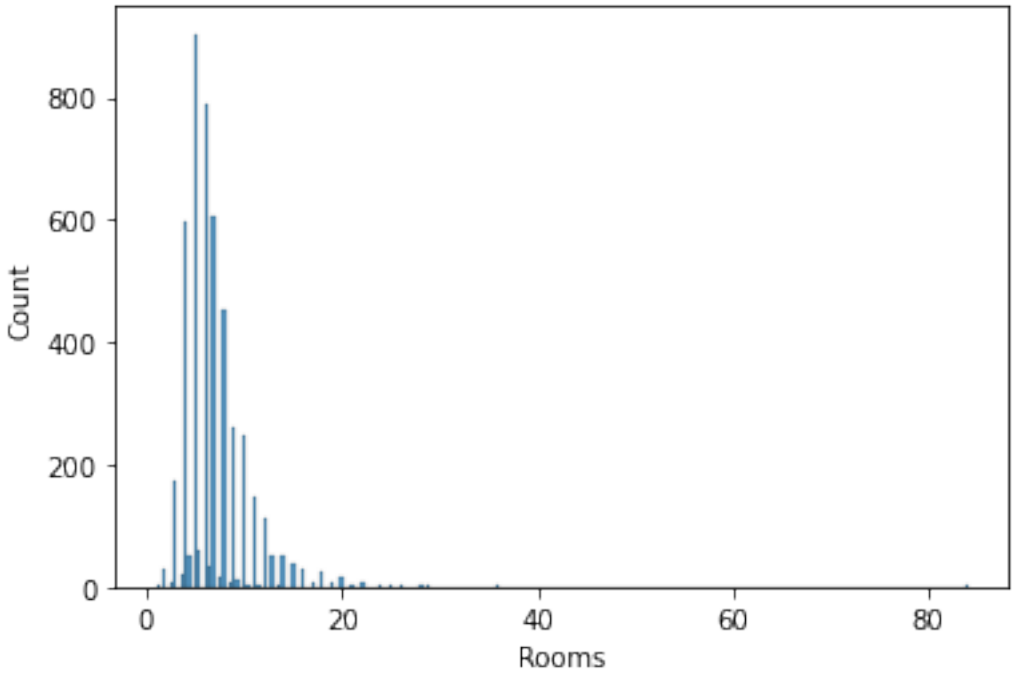
Lot size below 50 m^2 is very unlikely.

[31]:
```
data['Lot'] = np.log(data['Lot'].to_numpy())
data = data.loc[data['Lot'] > 4, :]
```

1.2.4 Rooms column

[32]:
```
sns.histplot(data['Rooms'])
```

[32]: <AxesSubplot:xlabel='Rooms', ylabel='Count'>



[33]:
```
np.count_nonzero((data['Rooms'] >= 30).to_numpy())
```

[33]:
```
10
```

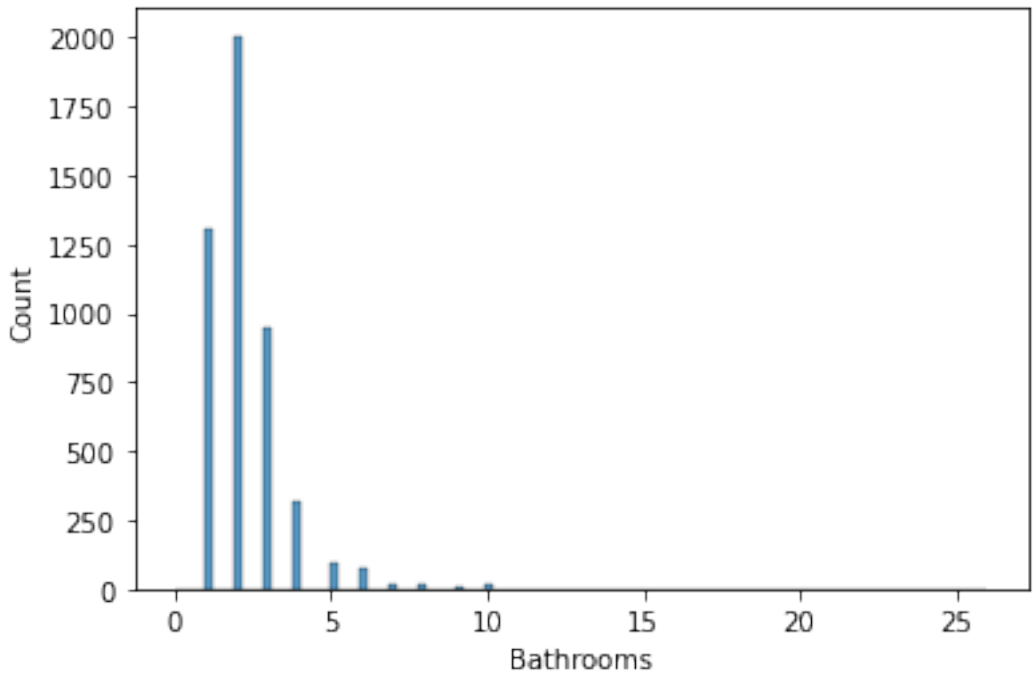
There are only very few sample with high number of rooms. There is no chance to get good predictions from those few samples.

[34]:
```
data = data.loc[data['Rooms'] < 30, :]
```

1.2.5 Bathrooms column

[35]:
```
sns.histplot(data['Bathrooms'])
```

[35]:
```
<AxesSubplot:xlabel='Bathrooms', ylabel='Count'>
```



[36]:
```
np.count_nonzero((data['Bathrooms'] >= 15).to_numpy())
```

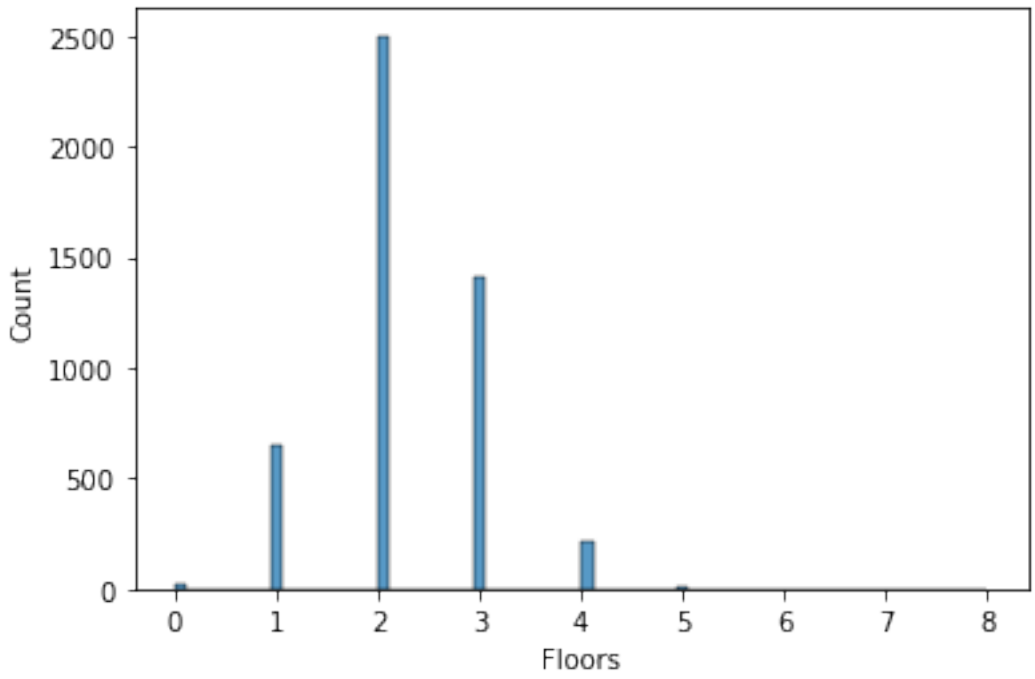
[36]: 4

[37]:
```
data = data.loc[data['Bathrooms'] < 15, :]
```

1.2.6 Floors column

[38]:
```
sns.histplot(data['Floors'])
```

[38]: <AxesSubplot:xlabel='Floors', ylabel='Count'>

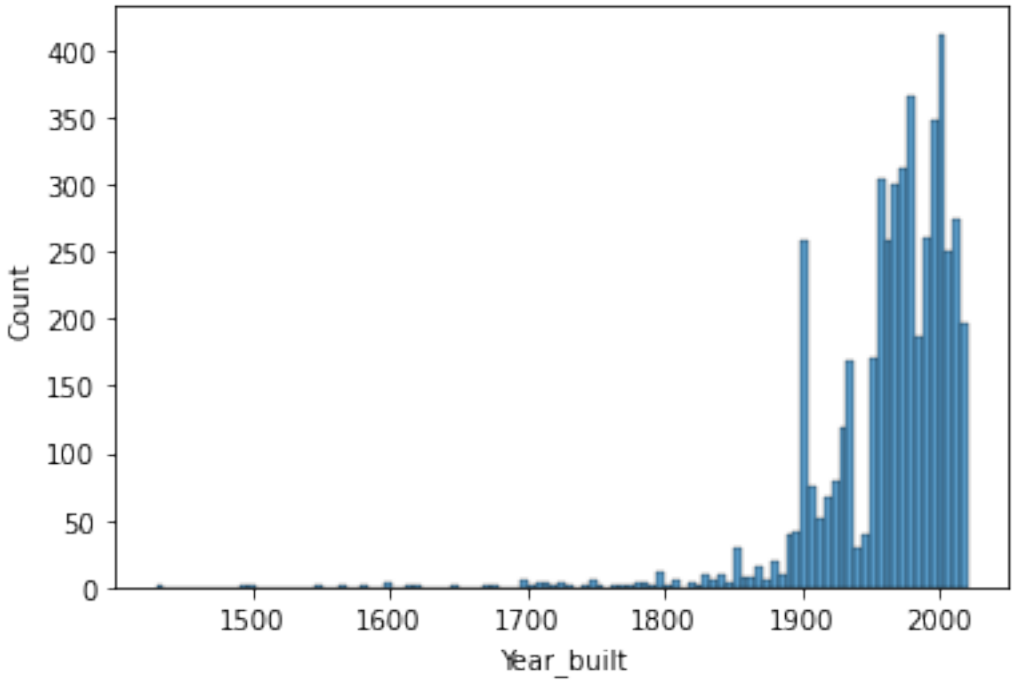


Nothing to do here.

1.2.7 `Year_built` column

[39]:
```
sns.histplot(data['Year_built'])
```

[39]:
```
<AxesSubplot:xlabel='Year_built', ylabel='Count'>
```



[40]:
```
np.count_nonzero((data['Year_built'] <= 1500).to_numpy())
```

[40]: 3

[41]:
```
data = data.loc[data['Year_built'] > 1500, :]
```

Values above 2020 obviously are wrong.

[42]:
```
np.count_nonzero((data['Year_built'] > 2020).to_numpy())
```

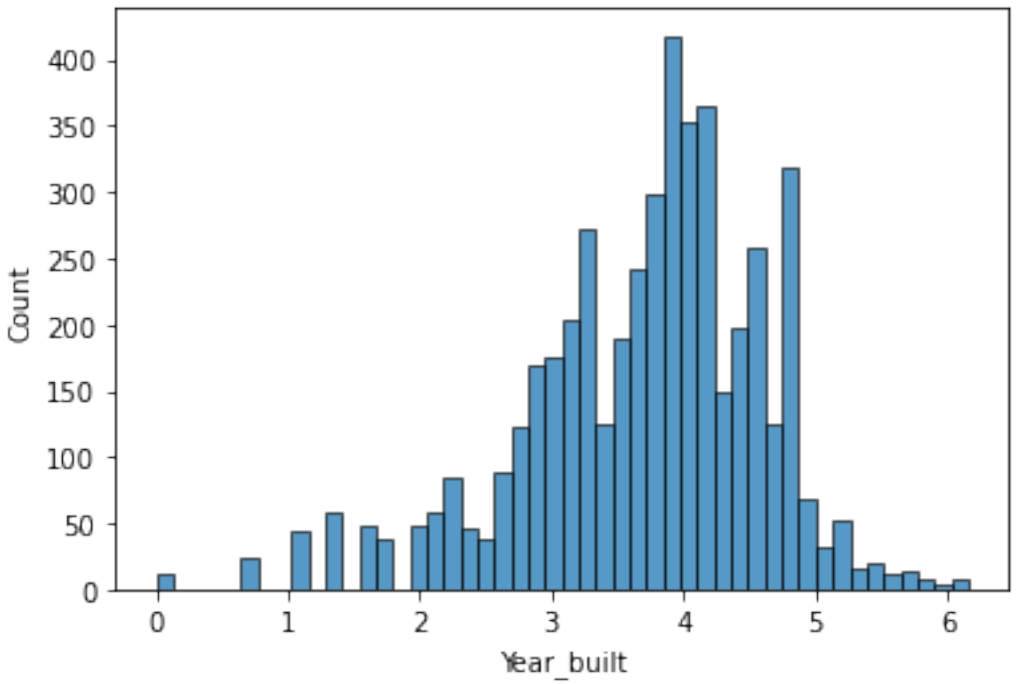
[42]: 6

[43]:
```
data = data.loc[data['Year_built'] <= 2020, :]
```

To get a better distribution of the samples over the range, we again apply a logarithmic transform.

[44]:
```
data['Year_built'] = np.log(2021 - data['Year_built'].to_numpy())
sns.histplot(data['Year_built'])
```

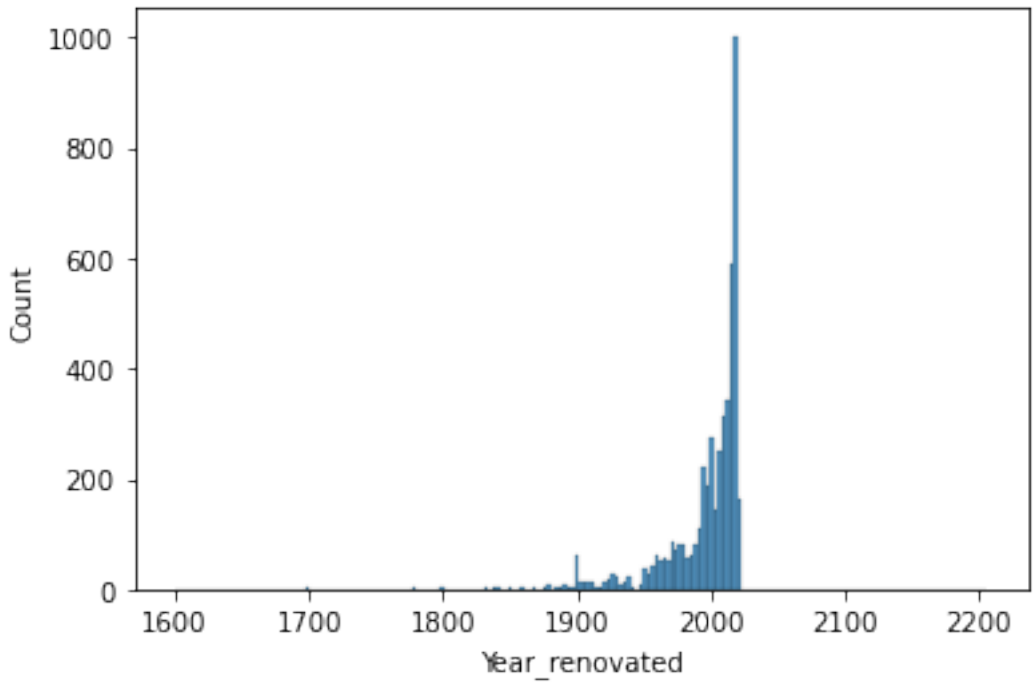
[44]: <AxesSubplot:xlabel='Year_built', ylabel='Count'>



1.2.8 Year_renovated column

[45]:
```
sns.histplot(data['Year_renovated'])
```

[45]:
```
<AxesSubplot:xlabel='Year_renovated', ylabel='Count'>
```



There seem to be renovations before 1900, which seems somewhat strange. But remember that we filled missing values with values from `Year_built`. Values above 2020 obviously are wrong.

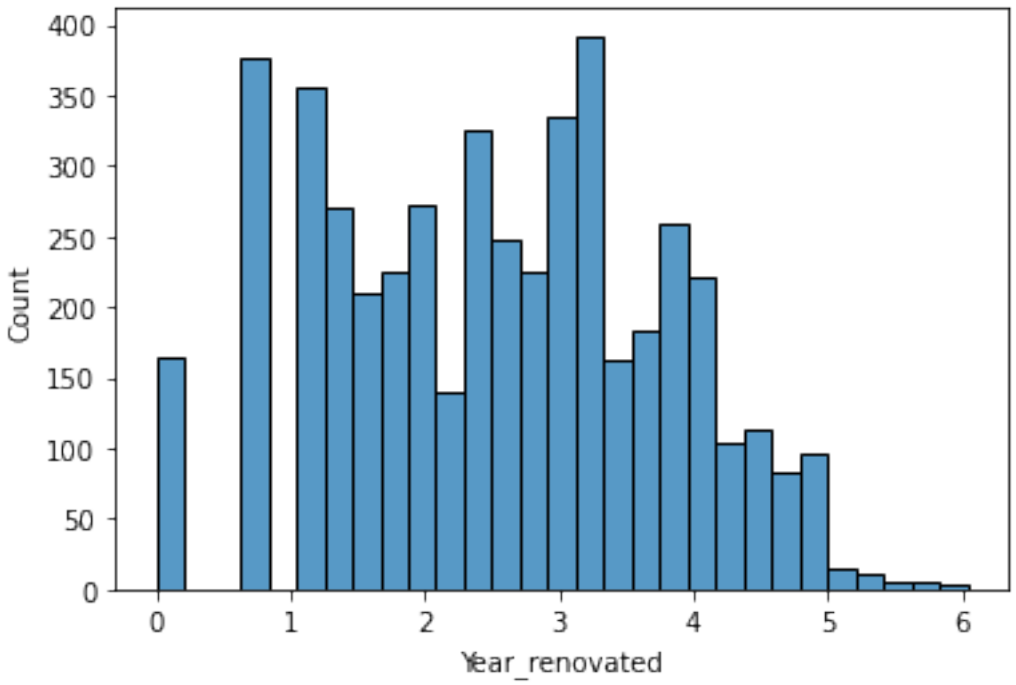
[46]:
```
np.count_nonzero((data['Year_renovated'] > 2020).to_numpy())
```

[46]:
```
2
```

[47]:
```
data = data.loc[data['Year_renovated'] <= 2020, :]
```

[48]:
```
data['Year_renovated'] = np.log(2021 - data['Year_renovated'].to_numpy())
sns.histplot(data['Year_renovated'])
```

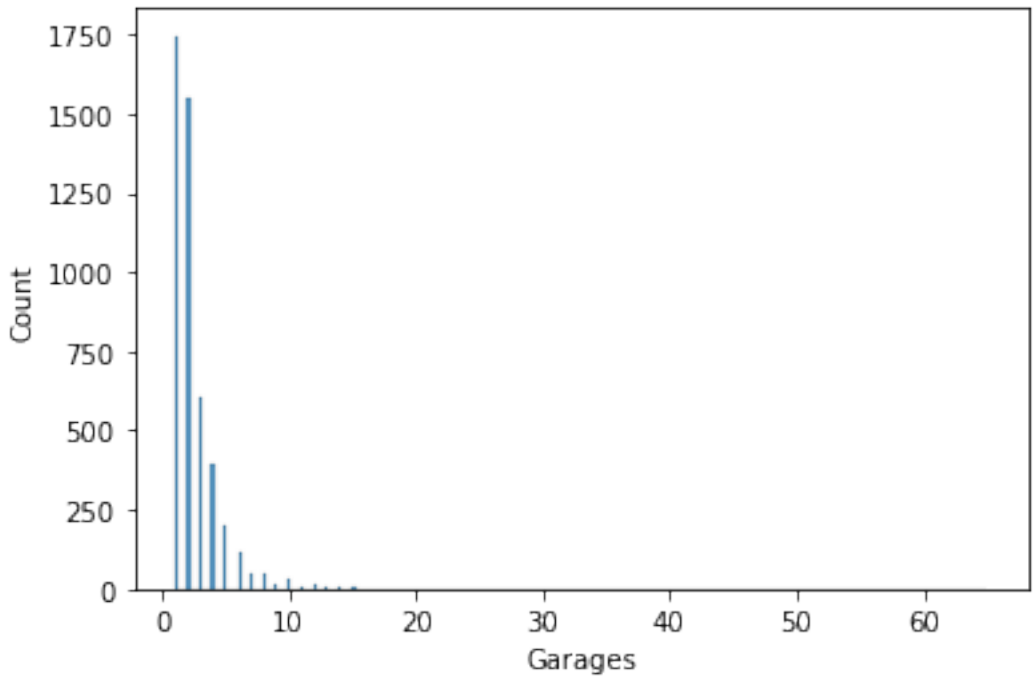
[48]:
```
<AxesSubplot:xlabel='Year_renovated', ylabel='Count'>
```



1.2.9 Garages column

[49]:
```
sns.histplot(data['Garages'])
```

[49]:
```
<AxesSubplot:xlabel='Garages', ylabel='Count'>
```



[50]:
```
np.count_nonzero((data['Garages'] >= 20).to_numpy())
```

[50]:
```
9
```

[51]:
```
data = data.loc[data['Garages'] < 20, :]
```

1.2.10 Type column

[52]:
```
data.groupby('Type').count()['Price'].sort_values()
```

[52]:
```
Type
Castle                    2
Corner house             78
Bungalow                113
Residential property    122
Special property        168
Multiple dwelling       206
Villa                   213
Farmhouse               264
Single dwelling         565
Duplex                  868
Mid-terrace house      2195
Name: Price, dtype: int64
```

[53]:
```
data = data.loc[data['Type'] != 'Castle', :]
data['Type'].cat.remove_categories('Castle', inplace=True)
```

1.2.11 Condition column

[54]:
```
data.groupby('Condition').count()['Price']
```

[54]:
```
Condition
first occupation                          50
first occupation after refurbishment     209
as new                                     3
maintained                               361
renovated                                508
modernized                              2144
refurbished                              562
by arrangement                            26
fixer-upper                              268
dilapidated                              661
Name: Price, dtype: int64
```

We should remove `'as new'` and `'by arrangement'` because only few samples use these categories and both are somewhat dubious.

[55]:
```
data = data.loc[~data['Condition'].isin(['as new', 'by arrangement']), :]
data['Condition'].cat.remove_categories(['as new', 'by arrangement'],␣
↪inplace=True)
```

1.2.12 Heating column

[56]:
```
data.groupby('Heating').count()['Price'].sort_values()
```

[56]:
```
Heating
solar heating              10
cogeneration units         13
gas heating                32
electric heating           52
floor heating              55
wood-pellet heating        61
district heating          117
night storage heater      136
underfloor heating        162
central heating           248
oil heating               418
heat pump                 562
stove heating            2897
Name: Price, dtype: int64
```

Something is wrong here! More than every second house sold in 2020 has stove heating? And what

about 'floor heating'? Is it gas powered or oil powered or what else? What's the difference between 'floor heating' and 'underfloor heating'. It's better to drop this column.

[57]:
```
data = data.drop(columns=['Heating'])
```

1.2.13 Garage_type column

[58]:
```
data.groupby('Garage_type').count()['Price'].sort_values()
```

[58]:
```
Garage_type
Car park lot                  1
Duplex lot                   26
Underground parking lot      52
Carport                     409
Parking lot                 739
Outside parking lot         891
Garage                     2645
Name: Price, dtype: int64
```

There are many similar categories. We should join some.

[59]:
```
data.loc[data['Garage_type'] == 'Car park lot', 'Garage_type'] = 'Outside␣
↪parking lot'
data.loc[data['Garage_type'] == 'Duplex lot', 'Garage_type'] = 'Outside parking␣
↪lot'
data.loc[data['Garage_type'] == 'Parking lot', 'Garage_type'] = 'Outside␣
↪parking lot'
data['Garage_type'].cat.remove_categories(['Car park lot', 'Duplex lot',␣
↪'Parking lot'], inplace=True)

data.groupby('Garage_type').count()['Price'].sort_values()
```

[59]:
```
Garage_type
Underground parking lot      52
Carport                     409
Outside parking lot        1657
Garage                     2645
Name: Price, dtype: int64
```

1.3 Save cleaned data

We save cleaned data for future use.

[60]:
```
data.to_csv(data_path + 'german_housing_preprocessed.csv')
```

1.4 Linear regression

Now data is almost ready for training a model. It remains to convert categorical data to numerical data. Condition is ordered and numeric representation is accessible with Series.cat.codes.

Columns `Type` and `Garage_type` should be one-hot encoded.

[61]:
```
data['Condition_codes'] = data['Condition'].cat.codes
data = pd.get_dummies(data, columns=['Type', 'Garage_type'], drop_first=True)
```

[62]:
```
data.head()
```

[62]:

	Price	Living_space	Lot	Rooms	Bathrooms	Floors	Year_built
0	13.118355	4.663439	5.433722	5.5	1.0	2.0	2.772589
2	13.526494	5.093075	4.406719	5.0	2.0	4.0	2.079442
3	12.464583	4.941642	6.701960	4.0	2.0	2.0	4.795791
8	12.804909	5.424950	6.880384	10.0	4.0	2.0	5.356586
10	14.375126	5.347108	7.286192	6.0	2.0	3.0	4.406719

	Year_renovated	Condition	State	City
0	2.772589	modernized	Baden-Württemberg	Bodenseekreis
2	2.079442	dilapidated	Baden-Württemberg	Esslingen (Kreis)
3	3.044522	fixer-upper	Baden-Württemberg	Waldshut (Kreis)
8	1.791759	modernized	Baden-Württemberg	Enzkreis
10	1.945910	modernized	Baden-Württemberg	Stuttgart

	Place	Garages	Condition_codes	Type_Corner house
0	Bermatingen	2.0	4	0
2	Ostfildern	1.0	7	0
3	Bonndorf im Schwarzwald	1.0	6	0
8	Neuenbürg	8.0	4	0
10	Schönberg	2.0	4	0

	Type_Duplex	Type_Farmhouse	Type_Mid-terrace house
0	0	0	0
2	0	1	0
3	0	1	0
8	1	0	0
10	0	0	1

	Type_Multiple dwelling	Type_Residential property	Type_Single dwelling
0	1	0	0
2	0	0	0
3	0	0	0
8	0	0	0
10	0	0	0

	Type_Special property	Type_Villa	Garage_type_Garage
0	0	0	0
2	0	0	1
3	0	0	1
8	0	0	0

```
10                      0           0                    1

    Garage_type_Outside parking lot  Garage_type_Underground parking lot
0                                  1                                   0
2                                  0                                   0
3                                  0                                   0
8                                  1                                   0
10                                 0                                   0
```

We drop columns not used for regression and convert the data frame to NumPy arrays suitable for Scikit-Learn.

[63]:
```
y = data['Price'].to_numpy()
X = data.drop(columns=['Price', 'Condition', 'State', 'City', 'Place']).
 ↪to_numpy()

print(X.shape, y.shape)
```

```
(4763, 21) (4763,)
```

We relatively few data. Thus, test set should to be small to have more training samples.

[64]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,␣
 ↪test_size=0.2)
print(y_train.size, y_test.size)
```

```
3810 953
```

We use polynomial regression with regularization.

[65]:
```
steps = [('poly', preprocessing.PolynomialFeatures()),
         ('ridge', linear_model.Ridge())]

pipe = pipeline.Pipeline(steps)

param_grid = {'poly__degree': [1, 2, 3],
              'ridge__alpha': [0] + [2 ** k for k in range(5, 15)]}

gs = model_selection.GridSearchCV(pipe, param_grid=param_grid,
                                  scoring='neg_mean_squared_error', n_jobs=-1,␣
 ↪cv=5)

gs.fit(X_train, y_train)
best_params = gs.best_params_
```

1.5 Evaluating the model

Now we use the test set to evaluate prediction quality of the model.

[66]:
```
print(best_params)

pipe.set_params(**best_params)
pipe.fit(X_train, y_train)

y_test_pred = pipe.predict(X_test)
```

```
{'poly__degree': 2, 'ridge__alpha': 1024}
```

Root mean squared error between predicted and exact targets on its own does not tell much about fitting quality. We have to compare the value to standard deviation of the targets. Standard deviation is the root mean squared error of the exact targets and their mean. In other words, standard deviation tells us the prediction error if we would use constant predictions for all inputs. Obviously the constant should be the mean of the training (!) targets, but the mean of the training targets should be very close the mean of the test targets if test sample have been selected randomly.

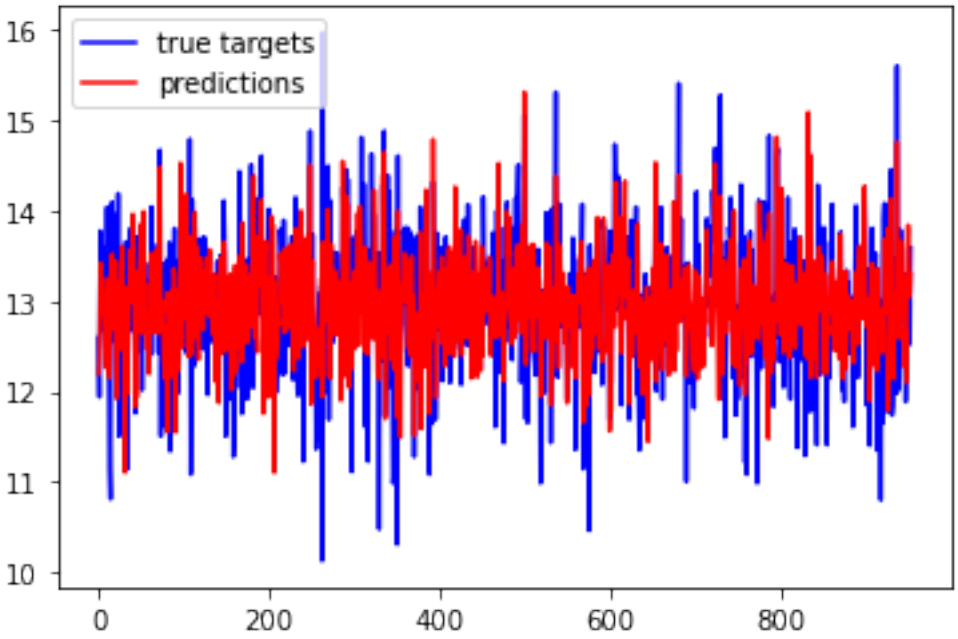
[67]:
```
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_test_pred))
sigma = np.std(y_test)
print('RMSE:', rmse)
print('standard deviation:', sigma)
print('ratio:', rmse / sigma)
```

```
RMSE: 0.5566483082993788
standard deviation: 0.7612806510260215
ratio: 0.7311998637416465
```

We see that the model's prediction is better than constant prediction, but not so much.

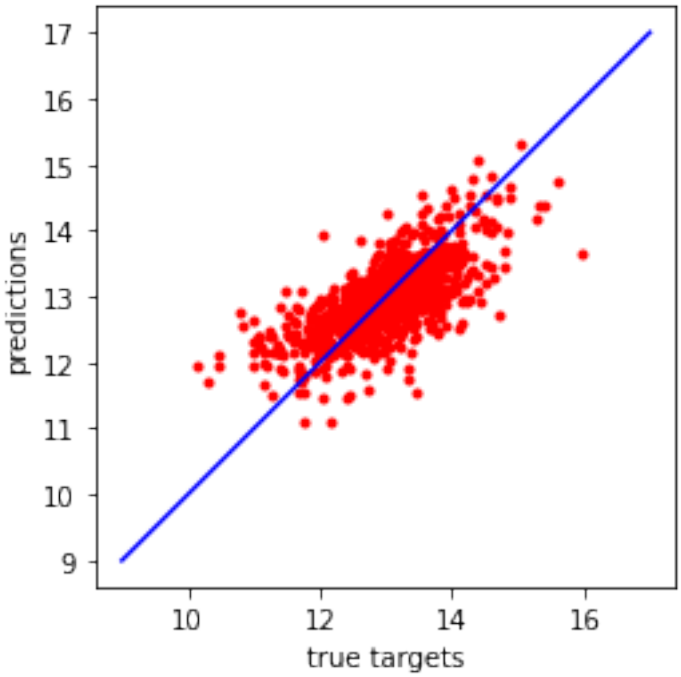
We should have a closer look at the predictions. Since there is no natural ordering in the set of samples plotting `y_test` and `y_test_pred` with `plot` does not help much.

[68]:
```
fig, ax = plt.subplots()
ax.plot(y_test, '-b', label='true targets')
ax.plot(y_test_pred, '-r', label='predictions')
ax.legend()
plt.show()
```



A better idea is to plot `y_test` versus `y_test_pred`. If true and predicted labels are close, then points should concentrate along the diagonal. Else they are far away from the diagonal.

[69]:
```
fig, ax = plt.subplots()
ax.plot(y_test, y_test_pred, 'or', markersize=3)
ax.plot([9, 17], [9, 17], '-b')
ax.set_xlabel('true targets')
ax.set_ylabel('predictions')
ax.set_aspect('equal')
plt.show()
```

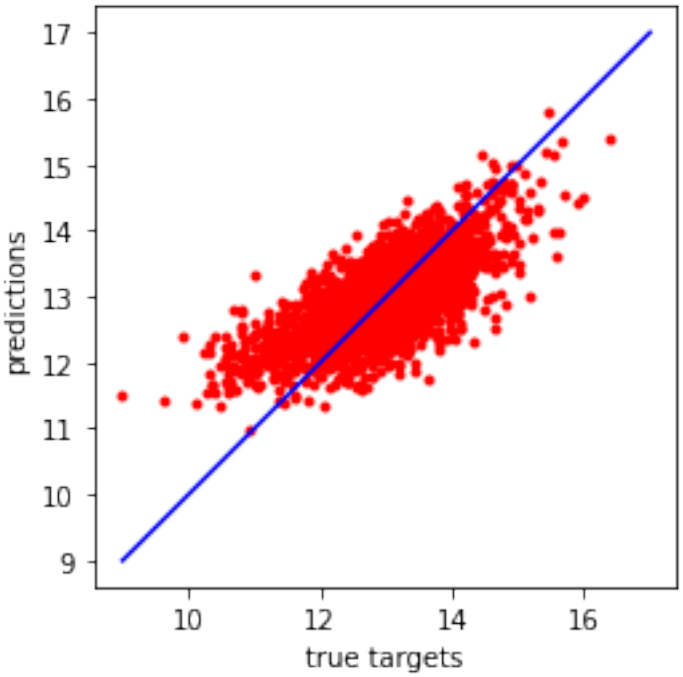


The red cloud shows some rotation compared to the blue line. Small target values get too high predictions and high target values get too low predictions. In other words, predictions tend to be too close to the target's mean. Such behavior is typically observed if there are man similar samples with different targets in the training data. Then there is no clear functional dependence of the targets on the inputs and models tend to predict the mean targets.

To further investigate this issue we should look at the predictions on the training set. If we are right, then predictions on the training set should show similar behavior (predictions close to mean).

[70]:
```
y_train_pred = pipe.predict(X_train)

fig, ax = plt.subplots()
ax.plot(y_train, y_train_pred, 'or', markersize=3)
ax.plot([9, 17], [9, 17], '-b')
ax.set_xlabel('true targets')
ax.set_ylabel('predictions')
ax.set_aspect('equal')
plt.show()
```



Again we see slight rotation. To summarize: our input data has too few details to explain the targets. There are simlar inputs with different targets leading to underestimation of high values and over estimation of low values. The only way out is gathering more data, either by dropping less columns or by getting relevant data from additional sources. We will come back to this issue soon.

1.6 Predictions

When using our model for predicting house prices we have to keep in mind that we transformed some of the input data. All those transforms have to be applied to new inputs, too.

[71]:
```
living_space = 80
lot = 3600
rooms = 5
bathrooms = 0
floors = 2
year_built = 1948
year_renovated = 1948
garages = 2
condition_codes = 7    # 0 = 'first occupation', 7 = 'dilapidated'
type_corner_house = 0
type_duplex = 0
type_farmhouse = 1
type_midterrace_house = 0
type_multiple_dwelling = 0
```

```
type_residential_property = 0
type_single_dwelling = 0
type_special_property = 0
type_villa = 0
garage_type_garage = 0
garage_type_outside_parking_lot = 1
garage_type_underground_parking_lot = 0

X = np.asarray([np.log(living_space), np.log(lot), rooms, bathrooms, floors,
                np.log(2021 - year_built), np.log(2021 - year_renovated),
                garages, condition_codes, type_corner_house, type_duplex,␣
 ↪type_farmhouse,
                type_midterrace_house, type_multiple_dwelling,␣
 ↪type_residential_property,
                type_single_dwelling, type_special_property, type_villa,
                garage_type_garage, garage_type_outside_parking_lot,
                garage_type_underground_parking_lot]).reshape(1, -1)

y = np.exp(pipe.predict(X))

print('predicted price: {:.0f} EUR'.format(y[0]))
```

```
predicted price: 128205 EUR
```

[ ]: